

TD 1h — Installer un projet Python, et décrire son installation

Guide détaillé, étape par étape

Le TD part d'un petit projet Python, recette, qui lit une recette écrite en Markdown et un tableau d'ingrédients, adapte les quantités au nombre de convives et écrit une page HTML. Le projet est complet, à une exception près : aucun de ses fichiers n'indique comment l'installer. On l'installe d'abord à la main dans un environnement conda neuf, en observant ce qui manque à chaque étape, puis on écrit les deux fichiers qui décrivent cette installation, et on les vérifie en refaisant l'environnement à partir d'eux. Le TD dure une vingtaine de minutes.

Le TD nécessite VS Code, avec le terminal réglé au TD 1c pour fonctionner comme une invite de commandes d'Anaconda, et un navigateur. Les étapes 2, 3 et 5 téléchargent des paquets : la session réseau doit être ouverte.

Étape	Objectif	Ce qu'on fait
1	lire un projet avant de l'installer	copier le projet, l'ouvrir dans VS Code, et le lire
2	partir d'un environnement qui ne contient que Python	créer un environnement d'essai qui ne contient que Python
3	trouver et installer ce dont le programme dépend	installer les deux sortes de dépendances, et lancer le programme
4	décrire l'environnement et l'installation du projet	écrire <code>environment.yml</code> et la section « Installation » du README
5	refaire l'environnement à partir de sa seule description	effacer l'environnement, et le refaire en suivant ces deux fichiers

La feuille du TD numérote ses opérations de 1 à 11 ; le guide reprend ces numéros. Chaque étape commence par un encadré qui la résume. Ce que chaque étape fait constater est expliqué à la fin du guide, dans « Ce que le TD fait constater » : faire l'étape d'abord, et noter ce qu'on observe, avant de lire l'explication.

1 · Copier le projet, l'ouvrir, et le lire

À faire : copier `depart\recette\` dans `travail\` ; ouvrir `travail\recette\` dans VS Code ; ouvrir un terminal ; dire à quoi sert chaque fichier du projet avant d'en lancer un.

À obtenir : l'explorateur de VS Code montre `pyproject.toml`, `README.md`, `data`, `src` et les deux programmes ; l'invite du terminal commence par `(base)` et se termine par `travail\recette>`.

Le dossier du TD

Ouvrir `info01\cours1\1h_recette\` dans l'explorateur de fichiers. Le dossier contient :

```
1h_recette\  
├─ depart\  
│   └─ recette\      le projet, tel qu'il est fourni  
├─ travail\         vide  
├─ td_1h_recette.pdf la feuille du TD  
└─ README.md
```

Comme dans les TD précédents, `depart\` contient les fichiers fournis, et ne se modifie pas. Copier le dossier `recette` : sélectionner `depart\recette`, `Ctrl + C`, ouvrir `travail\`, `Ctrl + V`. La copie, `travail\recette\`, est le dossier sur lequel porte tout le reste du TD. Son chemin complet est :

```
C:\Users\eleve\Desktop\info01\cours1\1h_recette\travail\recette
```

Ouvrir le projet dans VS Code

1. Lancer VS Code comme au TD 1c.
2. Menu File, Open Folder (Fichier, Ouvrir le dossier), et choisir `travail\recette\`. Ouvrir le dossier du projet entier, et non un fichier seul.
3. À la question sur la confiance accordée aux auteurs, répondre « Yes, I trust the authors ». Sans cette réponse, l'extension Python ne se charge pas.
4. Menu Terminal, New Terminal. Le terminal s'ouvre en bas de la fenêtre, déjà placé dans le dossier du projet.

Vérification : l'invite du terminal a la forme

```
(base) C:\Users\eleve\Desktop\info01\cours1\1h_recette\travail\recette>
```

`(base)` est l'environnement actif, et le `>` final indique un terminal cmd. Si l'invite commence par `PS` et qu'un message `activate.ps1 cannot be loaded` s'affiche, le terminal est un PowerShell : faire le réglage du terminal du TD 1c, puis ouvrir un nouveau terminal. À défaut, ouvrir l'invite de commandes d'Anaconda (« Anaconda Prompt » dans le menu Démarrer), taper `cd` suivi d'une espace, glisser le dossier `travail\recette` dans la fenêtre, ce qui écrit son chemin, puis Entrée. Toutes les commandes du TD se tapent dans ce terminal.

Ce que le projet contient

Le projet contient :

```
recette\  
├─ pyproject.toml  
├─ README.md  
├─ data\  
│   ├── ingredients.csv  
│   ├── recette.md  
│   └─ style.css  
├─ src\  
│   └─ recette\  
│       └─ __init__.py
```

```

|   ├── __main__.py
|   ├── calculs.py
|   └── tableau.py
└── recette_a_la_main.py
    └── recette_avec_tabulate.py

```

Ouvrir chaque fichier d'un clic dans l'explorateur de VS Code, sans rien lancer, et remplir le tableau de la feuille :

Ce que le projet contient	Ce que c'est
<code>pyproject.toml</code>	
<code>data\</code>	
<code>src\recette\</code>	
<code>recette_a_la_main.py</code> , <code>recette_avec_tabulate.py</code>	
<code>README.md</code>	

Pour les deux programmes, comparer leurs lignes `import`, en haut du fichier. Pour `pyproject.toml`, lire la ligne `dependencies`. Le `README` s'affiche mis en forme avec `Ctrl + K` puis `V`, comme au TD 1f.

À noter : ce qui manque au projet pour qu'une autre personne puisse l'installer. Le premier paragraphe du `README` du projet l'indique.

Les noms importés

Les deux programmes importent quatre noms. `pathlib` fait partie de Python ; les trois autres n'en font pas partie, et n'ont pas la même provenance :

Dans <code>recette_a_la_main.py</code>	Provenance
<code>import markdown</code>	la bibliothèque <code>markdown</code> , publiée sur les dépôts de paquets
<code>from recette.calculs import ...</code> , <code>from recette.tableau import ...</code>	le dossier <code>src\recette\</code> du projet lui-même

`recette_avec_tabulate.py` importe en plus `tabulate`, une bibliothèque publiée comme `markdown`. Ces deux sortes de dépendances s'installent par deux commandes différentes, à l'étape 3.

2 · Créer un environnement d'essai

À faire : créer l'environnement recette, qui ne contient que Python ; l'activer ; lister ce qu'il contient ; lancer `recette_a_la_main.py`.

À obtenir : l'invite commence par (recette) ; le programme s'arrête sur un `ModuleNotFoundError`.

Créer et activer l'environnement

1. Dans le terminal, taper :

```
conda create -n recette -c conda-forge python=3.12
```

-n recette donne le nom de l'environnement, -c conda-forge le dépôt d'où viennent les paquets, et python=3.12 la version de Python. conda calcule ce qu'il faut installer, en affiche la liste, puis demande Proceed ([y]/n)? : taper y puis Entrée. La commande se termine par les lignes qui indiquent comment activer l'environnement :

```
# To activate this environment, use
#
#     $ conda activate recette
```

2. Taper :

```
conda activate recette
conda list
```

conda list affiche un paquet par ligne, après trois lignes d'en-tête qui commencent par #.

Vérification : l'invite commence par (recette) au lieu de (base).

À noter : le nombre de paquets de l'environnement, et si markdown, tabulate ou recette sont dans la liste. Noter aussi quelques noms de la liste qui ne sont pas python.

Voir l'environnement dans VS Code

L'environnement actif se lit à deux endroits : en tête de l'invite du terminal, et en bas à droite de la fenêtre, dans la barre d'état, où l'extension Python affiche l'interpréteur choisi pour l'éditeur. Pour que l'éditeur emploie le nouvel environnement :

- Ctrl + Maj + P, taper « Python: Select Interpreter », Entrée ;
- choisir la ligne recette, dont le chemin contient .conda\envs\recette ;
- ouvrir un nouveau terminal (menu Terminal, New Terminal).

Le choix de l'interpréteur s'applique au bouton d'exécution, au débogueur et aux terminaux ouverts ensuite ; un terminal déjà ouvert conserve son environnement. Si recette n'est pas proposé, VS Code n'a pas relu la liste des environnements : palette, « Python: Clear Cache and Reload Window », puis recommencer.

Vérification : la barre d'état et l'invite du terminal nomment toutes les deux recette.

Lancer le programme

3. Taper :

```
python recette_a_la_main.py
```

Si le terminal n'est pas dans le dossier du projet (invite de commandes d'Anaconda ouverte depuis le menu Démarrer), s'y placer d'abord avec `cd C:\Users\eleve\Desktop\info01\cours1\1h_recette\travail\recette`.

À noter : la dernière ligne du message d'erreur, et le nom qu'elle cite.

Erreurs fréquentes

- 'conda' n'est pas reconnu en tant que commande interne ou externe : le terminal n'est pas une invite de commandes d'Anaconda. Refaire le réglage du TD 1c, ou ouvrir une invite de commandes d'Anaconda depuis le menu Démarrer.
- CondaToSNonInteractiveError: Terms of Service have not been accepted, ou la question Do you accept the Terms of Service (ToS) ... [(a)ccept/(r)ect/(v)iew] : conda demande, une fois par compte, d'accepter les conditions d'utilisation des canaux d'Anaconda. Répondre a, ou relancer la commande en ajoutant --override-channels avant -c conda-forge, ce qui n'emploie que conda-forge.
- CondaHTTPError: HTTP 000 CONNECTION FAILED : la session réseau n'est pas ouverte. L'ouvrir, puis relancer la commande. Une ligne « Solving environment » qui dure plusieurs minutes, sans erreur, est normale la première fois : conda télécharge la liste des paquets de conda-forge.
- CondaValueError: prefix already exists : un environnement recette existe déjà sur ce poste, créé lors d'un essai précédent. Le supprimer comme à l'étape 5 (conda env remove -n recette), puis recommencer.

3 · Résoudre les deux dépendances

À faire : installer markdown et tabulate avec conda ; relancer le programme ; installer le projet lui-même avec pip ; relancer le programme ; ouvrir la page produite.

À obtenir : le terminal affiche `recette.html` écrit pour 4 personne(s), en unités SI ; le navigateur affiche la recette mise en page.

Avant chaque commande d'installation, relire l'invite : elle doit commencer par `(recette)`. Une installation lancée depuis `(base)` s'applique à l'environnement de base d'Anaconda, que le compte élève ne peut pas modifier, et conda répond `EnvironmentNotWritableError`. Dans ce cas, taper `conda activate recette` et relancer.

Les bibliothèques publiées

4. Taper :

```
conda install -c conda-forge markdown tabulate
```

conda affiche la liste des paquets qu'il va installer, sous `The following NEW packages will be INSTALLED:`, puis demande `Proceed ([y]/n)?` : taper `y` puis Entrée.

À noter : le nombre de paquets de la liste, et leurs noms, comparés aux deux noms demandés.

5. Relancer le programme :

```
python recette_a_la_main.py
```

À noter : le nom cité par le message d'erreur, comparé à celui de l'étape 2, et d'où vient ce nom (voir « Les noms importés », à l'étape 1).

Le projet lui-même

6. Taper, depuis le dossier du projet :

```
python -m pip install -e .
```

Le point final fait partie de la commande : il désigne le dossier courant, c'est-à-dire le projet. `pip` lit `pyproject.toml`, vérifie les dépendances qui y sont déclarées, et installe le projet. Il télécharge aussi, pour la durée de l'installation, l'outil qui fabrique le paquet (`setuptools`, déclaré dans `[build-system]`). La sortie se termine par :

```
Successfully installed recette-0.1.0
```

À noter : ce que `pip` écrit sur `markdown` et `tabulate` (lignes `Requirement already satisfied`), et le dossier apparu dans `src\`.

Le programme, et la page produite

7. Relancer le programme :

```
python recette_a_la_main.py
```

Il affiche :

```
recette.html écrit pour 4 personne(s), en unités SI
```

Le fichier `recette.html` apparaît à la racine du projet, dans l'explorateur de VS Code. Pour l'ouvrir dans le navigateur : clic droit sur `recette.html`, « Reveal in File Explorer », puis double-clic sur le fichier dans l'explorateur de Windows.

Ensuite, dans VS Code, ouvrir `data\style.css`, remplacer la valeur d'une ligne `color:` ou `background:` (par exemple `background: #faf8f4;` en `background: #e8f0fa;`), enregistrer avec `Ctrl + S`, puis recharger la page dans le navigateur avec `F5`. Remettre la valeur d'origine ensuite.

Vérification : la page montre le titre « Crêpes », un tableau des ingrédients pour quatre personnes (240 g de farine, 500 ml de lait), puis les étapes de la préparation.

À noter : l'adresse de la page dans le navigateur, et ce qui change, dans la page et dans `recette.html`, quand on modifie `style.css`.

Sur un poste sans réseau, les opérations 1, 4 et 6, qui téléchargent des paquets, échouent. Passer alors à l'étape 4 du guide, qui ne demande que d'écrire des fichiers.

4 · Écrire ce qui manque

À faire : créer `environment.yml` à la racine du projet ; ajouter au `README.md` une section « Installation » qui donne les commandes, dans l'ordre.

À obtenir : `environment.yml` est à côté de `pyproject.toml` ; le `README` contient une section `## Installation` avec trois commandes.

Les quatre commandes des étapes 2 et 3 ne sont écrites nulle part dans le projet : une autre personne, ou soi-même sur un autre poste, devrait les retrouver de mémoire. Les deux fichiers écrits à cette étape les consignent.

Le fichier `environment.yml`

1. Dans l'explorateur de VS Code, cliquer sur une zone vide, sous les fichiers, pour que le nouveau fichier soit créé à la racine du projet et non dans `data\` ou `src\`.
2. Cliquer sur l'icône « New File... », en haut de l'explorateur, et taper `environment.yml`, puis Entrée.
3. Écrire le contenu suivant, puis enregistrer avec `Ctrl + S` :

```
name: recette
channels:
  - conda-forge
dependencies:
  - python=3.12
  - markdown
  - tabulate
```

`name` est le nom de l'environnement, `channels` le dépôt d'où viennent les paquets (le `-c conda-forge` des commandes), et `dependencies` la liste de ce qu'il faut installer. En YAML, l'indentation marque la structure : les lignes qui commencent par `-` sont décalées de deux espaces, et une tabulation est refusée. VS Code insère des espaces quand on appuie sur `Tab` dans un fichier `.yml`.

La section « Installation » du `README`

Ouvrir `README.md`. Après le paragraphe qui commence par « Il manque à ce projet deux choses », et avant la section `## Trois façons de s'en servir`, ajouter :

```
## Installation

```bash
conda env create -f environment.yml
conda activate recette
python -m pip install -e .
```
```

Les commandes se retrouvent dans l'historique du terminal : flèche vers le haut, dans le terminal, pour remonter les commandes tapées. Une phrase avant le bloc peut indiquer où les taper : dans un terminal où la commande `conda` est reconnue (une invite de commandes d'Anaconda sous Windows), depuis le dossier du projet.

Enregistrer, puis vérifier le rendu avec `Ctrl + K` puis `V`.

Vérification : l'explorateur de VS Code montre `environment.yml` au même niveau que `pyproject.toml` et `README.md`.

À noter : `pyproject.toml` cite déjà `markdown` et `tabulate` ; ce que chacun des deux fichiers décrit, et à quel outil il s'adresse. Noter aussi laquelle des trois commandes du `README` n'a pas d'équivalent dans `environment.yml`.

5 · Refaire l'environnement à partir du fichier

À faire : supprimer l'environnement recette ; le recréer avec `conda env create -f environment.yml` ; réinstaller le projet ; relancer le programme. Suivre pour cela uniquement la section « Installation » du README.

À obtenir : recette.html écrit pour 4 personne(s), en unités SI, sur un environnement refait de zéro.

Effacer l'environnement d'essai

8. Taper :

```
conda deactivate
conda env remove -n recette
```

`conda deactivate` fait sortir le terminal de l'environnement recette : l'invite commence de nouveau par (base). `conda` liste les paquets à supprimer, puis demande Proceed ([y]/n)? : taper y puis Entrée.

Vérification : `conda env list` ne cite plus recette, et le dossier du projet est intact.

Si `conda` répond Cannot remove current environment. Deactivate and run conda remove again, l'environnement est encore actif : taper `conda deactivate`, vérifier l'invite, et recommencer.

Le refaire en suivant le README

Ouvrir README.md et taper les commandes de la section « Installation », dans l'ordre, sans en ajouter.

9. Depuis le dossier du projet :

```
conda env create -f environment.yml
```

10. Puis :

```
conda activate recette
python -m pip install -e .
```

11. Enfin :

```
python recette_a_la_main.py
```

Vérification : le programme affiche recette.html écrit pour 4 personne(s), en unités SI, et la page s'ouvre comme à l'étape 3.

À noter : le nombre de commandes qu'il a fallu pour refaire l'environnement, comparé aux étapes 2 et 3, et d'où `conda` a pris le nom recette à l'opération 9. Si une commande a manqué, l'ajouter au README : une commande qu'il a fallu ajouter ici manque aussi à la documentation du projet.

Erreurs fréquentes à l'opération 9 :

- `EnvironmentFileNotFound` : le terminal n'est pas dans le dossier du projet, ou le fichier n'a pas été enregistré sous le nom `environment.yml`. Vérifier avec `dir` que le fichier est dans la liste.
- `CondaValueError: prefix already exists` : l'opération 8 n'a pas été faite, ou n'a pas abouti. Refaire l'opération 8.
- une erreur qui cite une ligne du fichier, avec le mot `yaml` : une indentation fausse ou une tabulation dans `environment.yml`. Comparer le fichier au modèle de l'étape 4.

Pour aller plus loin

- Depuis l'opération 10, la commande `recette` existe dans l'environnement. Taper `recette --help`, puis `recette --personnes 12 --unites US`, et recharger la page. `python -m recette` fait la même chose.

- Le README du projet demande aussi de documenter `recette_avec_tabulate.py` : écrire, à la place de la chaîne « À documenter », une chaîne sur le modèle de celle qui ouvre `recette_a_la_main.py`. Elle indique ce que le programme fait, comment on le lance, et ce qui le distingue de l'autre.
- Le même README propose une section « Exemples » : trois commandes et ce qu'elles produisent, dont une page écrite hors du dossier du projet (`recette --help` indique l'option à employer). Chaque commande écrite doit avoir été lancée.

En fin de séance, réactiver l'environnement de base dans le terminal avec `conda activate base`.

Ce que le TD fait constater

Cette section se lit après avoir fait les étapes.

Étape 1 : le contenu d'un projet Python

| Ce que le projet contient | Ce que c'est |
|--|---|
| pyproject.toml | le nom du projet, et ce dont son code dépend |
| data\ | ce que le programme lit, et la feuille de style |
| src\recette\ | le code, en trois modules |
| recette_a_la_main.py, recette_avec_tabulate.py | deux programmes qui ne diffèrent que par un import |
| README.md | ce que le projet fait, mais pas comment l'installer |

Une personne autre que l'auteur ne peut pas installer le projet : aucun fichier ne décrit l'environnement dans lequel le code s'exécute, et rien ne dit quelles commandes taper. Les deux fichiers de l'étape 4 comblent ces deux manques.

Les noms `data` et `src` ne sont pas imposés par Python. Ranger le code sous `src\` est la disposition la plus répandue, et la documentation de la Python Packaging Authority en donne la raison : le code n'est importable qu'une fois le projet installé, si bien qu'on travaille toujours sur la version installée, et jamais sur un dossier trouvé par hasard dans le dossier courant. L'opération 5 de la feuille fait constater cette règle.

Deux sortes de dépendances

| | Provenance | Commande d'installation |
|--------------------|----------------------|---|
| markdown, tabulate | du dépôt conda-forge | <code>conda install -c conda-forge markdown tabulate</code> |
| recette | du dossier du projet | <code>python -m pip install -e .</code> |

Une bibliothèque qu'on écrit soi-même n'est sur aucun dépôt tant que personne ne l'y a publiée, et c'est le cas ordinaire d'un projet de cours, d'un projet interne ou d'un projet en cours d'écriture. Elle s'installe depuis son dossier, que le point de la commande désigne. L'option `-e` (*editable*) installe le projet de sorte que Python lise le code directement dans le dossier du projet : une modification du code prend effet sans réinstallation.

Cette commande emploie `pip` plutôt que `conda`, parce que `conda` installe des paquets venus d'un dépôt, et ne sait pas installer un dossier de travail. C'est la seule exception à la règle d'un seul outil par environnement, et elle se limite au projet lui-même. `pip` est présent dans l'environnement dès sa création, sans qu'on l'ait demandé.

Étapes 2 et 3 : un environnement « Python seul »

| | Commande | Ce qu'on constate |
|---|--|--|
| 2 | <code>conda activate recette, puis conda list</code> | une vingtaine de paquets, et aucun des trois noms importés |
| 3 | <code>python recette_a_la_main.py</code> | <code>ModuleNotFoundError: No module named 'markdown'</code> |

La feuille donne 28 paquets, nombre relevé sous Linux ; sous Windows, la même commande en installait 20 en septembre 2026. Le nombre exact dépend du système et de la date ; dans tous les cas, un environnement qui ne contient que Python n'est pas vide. Il contient des bibliothèques écrites en C sans lesquelles l'interpréteur ne démarre pas, comme `openssl`, `libsqlite`, `libzlib`, et, sous Windows, les bibliothèques d'exécution de Microsoft (`ucrt`, `vc14_runtime`). `pip`, `setuptools` et `wheel` y sont aussi, d'où un `pip install` qui fonctionne dans un environnement `conda` sans qu'on l'ait installé.

Le programme s'arrête sur une dépendance manquante, alors que le fichier existe et que sa syntaxe est correcte. Le message cite `markdown` et non `tabulate` ou `recette`, parce que `import markdown` est le premier import du fichier qui échoue : Python s'arrête à la première ligne fautive, et ne fait pas la liste de ce qui manque.

Étape 3 : les deux dépendances résolues

| | Commande | Ce qu'on constate |
|---|---|---|
| 4 | <code>conda install -c conda-forge markdown tabulate</code> | quatre paquets : les deux demandés, <code>importlib-metadata</code> , <code>zipp</code> |
| 5 | <code>python recette_a_la_main.py</code> | <code>ModuleNotFoundError: No module named 'recette'</code> |
| 6 | <code>python -m pip install -e .</code> | le projet s'installe, ses dépendances étant déjà installées |
| 7 | <code>python recette_a_la_main.py</code> ,
puis ouvrir <code>recette.html</code> | la recette mise en page, sans serveur ni réseau |

À l'opération 4, `markdown` a lui-même des dépendances, `importlib-metadata` et `zipp`, que `conda` installe avec lui : ce sont des dépendances transitives du projet. Dans l'environnement `base`, la même commande en installerait moins, les autres y étant déjà.

Les messages des opérations 3 et 5 ont la même forme, et leurs causes diffèrent. `markdown` manquait à l'environnement, et `conda` l'a installé. `recette` est le code du projet, pourtant présent dans `src\recette` : Python ne le trouve pas, parce qu'il cherche les modules dans le dossier du programme lancé et dans ceux de l'environnement, et que `src` n'est ni l'un ni l'autre tant que le projet n'est pas installé.

À l'opération 6, `pip` lit `pyproject.toml`, constate que `markdown` et `tabulate` sont déjà installés (`Requirement already satisfied`), et ne les télécharge pas. Il crée `src\recette.egg-info`, qui décrit le projet installé ; ce dossier n'est pas versionné.

À l'opération 7, on retrouve trois notions vues pendant la séance. Le `recette.md` du projet ressemble à celui écrit au TD 1f. La page sépare le contenu, dans `recette.html`, de la présentation, dans `data\style.css`, comme les deux pages du poème au TD 1a : modifier la feuille de style change la page, et `recette.html` n'est pas modifié. Enfin, la page s'ouvre par une adresse `file:///`, sans serveur ni réseau.

Étape 4 : deux fichiers, deux rôles

`environment.yml` liste ce qu'il faut installer ; le `README` donne l'ordre des commandes, y compris celle que le fichier ne peut pas décrire, l'installation du projet lui-même par `pip`. Aucun des deux n'installe quoi que ce soit : ils décrivent une installation, que les commandes réalisent. Une installation décrite dans un fichier peut être refaite à l'identique, sans dépendre du souvenir des commandes tapées.

`pyproject.toml` déclare déjà `markdown` et `tabulate`, et les réécrire dans `environment.yml` n'est pas un doublon. Les deux fichiers ne s'adressent pas au même outil : `pyproject.toml` indique à `pip` de quoi le code a besoin, `environment.yml` indique à `conda` de quoi la machine a besoin, y compris ce qui n'est pas du Python, comme Python lui-même.

Étape 5 : la documentation vérifiée

| | Commande | Ce qu'on constate |
|----|--|---|
| 8 | <code>conda deactivate</code> , puis <code>conda env remove -n recette</code> | l'environnement d'essai est supprimé, le dossier du projet est conservé |
| 9 | <code>conda env create -f environment.yml</code> | les trois paquets demandés s'installent d'un coup, avec leurs dépendances |
| 10 | <code>conda activate recette</code> , puis <code>python -m pip install -e .</code> | le projet à nouveau importable |
| 11 | <code>python recette_a_la_main.py</code> | la même page, sur un environnement refait de zéro |

La suppression de l'opération 8 est nécessaire : le fichier porte name: `recette`, et `conda env create` refuse de créer un environnement dont le nom existe déjà. Recréer l'environnement à partir de rien vérifie que le fichier décrit l'environnement entier, et pas seulement ce qui manquait à l'ancien. `conda` refuse aussi de supprimer l'environnement actif, d'où le `conda deactivate` qui précède.

À l'opération 9, une seule commande remplace les opérations 1 et 4, et le nom de l'environnement n'est plus tapé : `conda` le lit dans la ligne name du fichier. Les opérations 9 et 10 sont les commandes de la section « Installation » : une documentation d'installation n'est vérifiée qu'une fois suivie par quelqu'un, sur un environnement qui ne contient rien.

Pour aller plus loin : la commande `recette`

La commande `recette` existe depuis l'opération 10, parce que la section `[project.scripts]` de `pyproject.toml` la déclare ; le cours 3 y revient. Elle équivaut à `python -m recette`, et accepte `--personnes`, `--unités` et `--sortie`. Une chaîne de documentation possible pour `recette_avec_tabulate.py` :

```
"""La recette pour quatre personnes, avec la bibliothèque `tabulate`.

    python recette_avec_tabulate.py

Le programme est celui de `recette_a_la_main.py`, à un import près : le
`tabulate` importé ici est celui de la bibliothèque du même nom, installée
dans l'environnement, et non celui écrit dans `src/recette/tableau.py`.
Pour un autre nombre de convives ou un autre système d'unités, changez
`PERSONNES` et `UNITES` et relancez.
"""
```